Electronic Version
Stylesheet Version v1.1.1

Description

USB Smart Switch with Packet Re-Ordering for Interleaving among Multiple Flash-Memory Endpoints Aggregated as a Single Virtual USB Endpoint

BACKGROUND OF INVENTION

[0001] This invention relates to Universal-Serial-Bus (USB) hubs, and more particularly to a UWB switch that aggregates endpoints into a single virtual endpoint.

[0002] Universal-Serial-Bus (USB) has become a popular standard interface for connecting peripherals to a host such as a personal computer (PC). USB flash-memory drives and devices have been developed to transport data from one host to another, replacing floppy disks. While large external flash drives may be used, smaller USB flash drives known as key-chain or key drives have been a rapidly growing market.

[0003] USB hubs allow one USB port on a host to fan out to multiple end USB devices or endpoints. A basic USB hub has a repeater that repeats data from the host to all downstream devices, while more intelligent hubs based on the USB 2.0 standard can buffer data to different downstream ports. This is especially useful when both faster and slower endpoint USB devices are connected to the same hub, since the slower hub's transaction can be buffered by the hub to allow a simultaneous transaction to the higher speed device. Software on the host can schedule transactions to different speed devices using split transaction commands to high-speed hubs. However, split transactions are not useful when all devices are already operating at the highest speed allowed by USB 2.0.

[0004] Whether transactions are buffered or not, all endpoint USB devices are visible to the host. The host can query each endpoint USB device for its status and can transfer data to or from each endpoint device. This is necessary when different kinds of endpoint devices are attached to a hub, such as a printer and a disk drive.

[0005] Figure 1 shows a prior-art USB hub that connects to multiple flash-memory USB endpoint devices. Host 10 includes USB host controller 12 that generates transactions

to USB devices over USB bus 18 using the USB protocol. USB hub 20 is connected to a cable containing USB bus 18. USB hub 20 fans out USB bus 18 to several downstream USB devices that connect over additional USB bus segments.

[0006] Three USB flash-memory systems 14, 15, 16 are connected to USB hub 20 by USB bus segments. USB flash-memory system 14 can be accessed by USB host controller 12 through USB hub 20 and appears to the user as drive E:, when host 10 is a PC that has a hard drive C: and an optical drive D: already installed. Since USB hub 20 passes all host transfers through to downstream devices, USB flash-memory system 15 is visible to host 10 as a second flash drive and is designated as drive F:, while USB flash-memory system 16 is visible to the host as a third flash drive designated as G: to the user.

[0007] As additional devices are added to host 10, either through USB hub 20 or through another interface, drives letters in ascending order are assigned to each new memory device. When many USB flash-memory systems such as flash cards are attached, the drive letters can reach the end of the alphabet, and even before then the user is presented with many drives to keep track of and to choose from

when storing or transferring data. This can be confusing and annoying to the user in much the same way that partitioned disk drives having multiple drive letters were.

[0008] What is desired is to aggregate multiple USB flash-memory systems together. It is desired to have all USB flash-memory systems appear as a single virtual USB endpoint so that the PC user sees the aggregated flash drives as a single drive letter. An intelligent USB hub that can act as a USB switch and aggregate multiple endpoints as a single USB for the host is desirable.

Brief Description of Drawings

[0009] Figure 1 shows a prior-art USB hub that connects to multiple flash-memory USB endpoint devices.

[0010] Figure 2 is a block diagram of a USB switch that aggregates and virtualizes multiple flash-memory endpoints.

[0011] Figure 3 shows the dual-mode USB switch in more detail.

[0012] Figure 4 is a packet-timing diagram of the USB switch operating in hub mode.

[0013] Figure 5 is a timing diagram of packet re-ordering by the USB switch in single-endpoint mode.

[0014] Figure 6 shows a packet being re-ordered between upstream and downstream queues.

[0015] Figure 7 is a table showing possible re-ordering packet sequences.

[0016] Figure 8 is a flowchart of packet re-ordering by the transaction manager.

[0017] Figure 9 is a flowchart of power-on initialization by the USB switch.

[0018] Figure 10 is a flowchart of high-level operation of the transaction manager.

[0019] Figure 11 is a flowchart of operation of the virtual storage processor.

[0020] Figure 12 shows a data access routine in a mirrored flash system.

[0021] Figure 13 is a flowchart of flash-memory access using data striping.

[0022] Figures 14A-C show various arrangements of data stored in the USB flash storage blocks.

DETAILED DESCRIPTION

[0023] The present invention relates to an improvement in USB hubs. The following description is presented to enable one of ordinary skill in the art to make and use the invention as provided in the context of a particular application and its requirements. Various modifications to the preferred embodiment will be apparent to those with skill in

the art, and the general principles defined herein may be applied to other embodiments. Therefore, the present invention is not intended to be limited to the particular embodiments shown and described, but is to be accorded the widest scope consistent with the principles and novel features herein disclosed.

[0024] Figure 2 is a block diagram of a USB switch that aggregates and virtualizes multiple flash-memory endpoints. USB host controller 12 for host 10 sends transactions over USB bus 18 to USB multi-flash device 40. USB switch 30 on USB multi-flash device 40 receives and responds to transaction from host 10 over USB bus 18.

[0025] Mode logic 26 causes USB switch 30 to operate in one of two modes. When mode pin 29 is grounded, mode logic 26 causes USB switch 30 to operate in a single-endpoint mode, where USB switch 30 aggregates all downstream USB flash-memory systems into a single USB endpoint that is visible host 10. The details about the number, size, speed, and arrangement of the physical USB flash-memory devices is hidden from USB host controller 12 by USB switch 30 when operating in single-endpoint mode. Host 10 sees a single pool of memory having one set of attributes. Attributes reported to host 10 can be chosen

by transaction manager 36, such as the slowest access time, or the sum of all the good blocks of memory.

[0026] When operating in single-endpoint mode, USB switch 30 acts as the final USB endpoint for transactions on USB bus 18 to host 10. USB switch 30 generates USB transactions on hidden USB buses 28 to USB flash storage blocks 22, 23, 24. USB flash storage blocks 22, 23, 24 respond to USB switch 30 over hidden USB buses 28 with USB switch 30 acting as the USB host on hidden USB buses 28. USB switch 30 then forwards data to host 10 by acting as the endpoint. Thus USB flash storage blocks 22, 23, 24 are hidden from host 10 when mode logic 26 activates the single-endpoint mode.

[0027] USB flash storage blocks 22, 23, 24 are aggregates together by USB switch 30, which maps and directs data transactions to selected USB flash storage blocks 22, 23, 24. Since USB switch 30 performs memory management, USB flash storage blocks 22, 23, 24 appear as a single, contiguous memory to host 10. Since host 10 sees USB switch 30 as the only endpoint of USB bus 18, data read or written to USB flash storage blocks 22, 23, 24 are all on a single virtual drive, such as drive letter E: on a PC. The details and complexities of USB flash storage blocks 22, 23,

24 are hidden from the end user.

[0028] When mode pin 29 is not grounded, mode logic 26 causes USB switch 30 to operate in multi-endpoint or hub mode. In hub mode, USB switch 30 acts as a normal USB hub, passing transactions from USB host controller 12 on USB bus 18 over hidden USB buses 28 to USB flash storage blocks 22, 23, 24. Host 10 then sees USB flash storage blocks 22, 23, 24 as the final USB endpoints. Each of the multiple flash endpoints appears as a different drive letter, E:, F:, G:, etc.

[0029] Figure 3 shows the dual-mode USB switch in more detail. USB switch 30 connects to host USB bus 18 through USB upstream interface 34. USB switch 30 connects to downstream USB flash storage blocks over hidden USB buses 28 through USB downstream interfaces 46, 47, 48. USB interfaces provide physical signaling, such as driving and receiving differential signals on differential data lines of USB buses, detecting or generating packet start or stop patterns, checking or generating checksums, and higher-level functions such as inserting or extracting USB device addresses and packet types and commands.

[0030] Mode logic 26 senses the voltage on mode pin 29, which can be pulled down to ground externally for single-

endpoint mode, or pulled high with a pull-up resistor for hub mode. Mode logic 26 activates USB switch 30 to operate as a hub or as an aggregating and virtualizing switch. For hub mode, data is buffered across virtual USB bridge 32 from the host to one of virtual USB bridges 42, 43, 44 to flash memory. Internal bus 38 allows data to flow among virtual USB bridge 32 and USB bridges 42, 43, 44. The host and the endpoint may operate at the same speed (USB low speed (LS), full speed (FS), or high-speed (HS)), or at different speeds. Buffers in virtual USB bridge 32 can store the data.

[0031] Virtual storage processor 140 provides re-mapping and translation services to transaction manager 36. For example, logical addresses from the host can be looked up and translated to physical device addresses in USB flash storage blocks 22, 23, 24.

[0032] When operating in single-endpoint mode, transaction manager 36 not only buffers data using virtual USB bridge 32, but can also re-order packets for transactions from the host. A transaction may have several packets, such as an initial token packet to start a memory read, a data packet from the memory device back to the host, and a handshake packet to end the transaction. Rather than

have all packets for a first transaction complete before the next transaction begins, packets for the next transaction can be re-ordered by USB switch 30 and sent to the memory devices before completion of the first transaction. This allows more time for memory access to occur for the next transaction. Transactions are thus overlapped by re-ordering packets.

[0033] Packets sent over hidden USB buses 28 are re-ordered relative to the packet order on host USB bus 18. Transaction manager 36 may overlap and interleave transactions to different USB flash storage blocks, allowing for improved data throughput. For example, packets for several incoming USB transactions from the host are stored in virtual USB bridge 32 or an associated buffer (not shown). Transaction manager 36 examines these buffered transactions and packets and re-orders the packets before sending them over internal bus 38 to a downstream USB flash storage block.

[0034] A packet to begin a memory read of a flash block through USB bridge 43 may be re-ordered ahead of a packet ending a read of another flash block through USB bridge 42 to allow access to begin earlier for the second flash block.

[0035] Figure 4 is a packet-timing diagram of the USB switch op-

erating in hub mode. USB software on host 10 schedules transactions to various USB devices for each time frame. Host 10 sends a token (non-data) packet to the flash memory device at endpoint 1. USB switch 30 passes this packet through from host 10 to endpoint-1, which is USB flash storage block 22. This token packet contains a write command to the flash block.

[0036] Next host 10 sends a data-out packet to endpoint-1, with the data to write into USB flash storage block 22. USB switch 30 acts as a hub and passes this data packet through. USB flash storage blocks 22 then writes this data into its flash memory and responds with a handshake packet back to host 10. USB switch 30 passes this handshake packet back to host 10.

[0037] Upon receiving the handshake packet from endpoint-1, host 10 then generates a second token packet that is sent to endpoint-2. USB switch 30 passes this second token packet through to USB flash storage block 23, the second USB endpoint that is addressed by the token packet. This token packet contains a flash-read command and an address to begin reading from and a length to read, so USB flash storage block 23 begins to read the requested data.

[0038] After a read access time, the data is ready to be sent back

to host 10. USB flash storage block 23 packs the data into a data-in packet. The data-in packet is sent to host 10 and passes through USB switch 30. A final handshake packet is also generated by USB flash storage block 23 to signal completion of the read command. The handshake packet is passed on to host 10 by USB switch 30.

[0039] Reading of flash data in USB flash storage block 23 cannot begin until time T2, since the second token packet is not sent by host 10 until after the first acknowledgement packet is received by the host. Thus initiation of data read is delayed until the prior transaction completes. The host could use a split transaction for the write, but since the host software lacks detailed knowledge of the endpoints, any packet re-ordering would be inefficient.

[0040] Figure 5 is a timing diagram of packet re-ordering by the USB switch in single-endpoint mode. USB switch 30 acts as the single endpoint seen by host 10. The first token packet with the command to write to USB flash storage is sent by host 10 and passed on to USB flash storage block 22. USB switch 30 determines which of USB flash storage blocks 22, 23, 24 to write, using a memory-mapping table of other memory-management techniques.

[0041] The data-out packet with the data to write from host 10 is

intercepted by USB switch 30 and the data is stored in a buffer. Instead of immediately sending the data to USB flash storage block 22, USB switch 30 generates a handshake packet back to host 10. Host 10 sees completion of the first transaction even though the data has not yet been sent to the physical flash device for storage.

[0042] Host 10 can then being the second transaction, sending the second token packet with a read command. USB switch 30 receives this packet, looks up the data's address, and determines that the data is stored in USB flash storage block 23. The second token packet is sent from USB switch 30 to USB flash storage block 23. This causes USB flash storage block 23 to begin reading the data at time T1.

[0043] USB switch 30 then generates a data-out packet to USB flash storage block 22, using the data stored earlier from the data-out packet from host 10. USB flash storage block 22 can then write this data to its flash memory upon receipt of the data-out packet. USB flash storage block 22 generates an acknowledgement or handshake packet to indicate completion of the write. This handshake packet is sent to USB switch 30, which closes the hidden transaction between USB switch 30 and USB flash storage block 22.

Host 10 already ended the transaction when USB switch 30 sent the initial handshake packet.

[0044] After a read-access time has elapsed, USB flash storage block 23 is ready to send the read data back to host 10. This data is packed in one or more data-in packets and sent to USB switch 30. USB switch 30 then forwards the data to host 10 in modified data-out packets that show USB switch 30 as the endpoint device. USB switch 30 can over-write the USB device address of USB flash storage block 23 with its own USB device address in the packet to make the USB data-in packet appear to host 10 as being from USB switch 30 rather than from hidden USB flash storage block 23.

[0045] A final handshake packet is generated by USB flash storage block 23 and sent to USB switch 30 to indicate completion of the read operation. USB switch 30 over-writes the USB device address of USB flash storage block 23 with the USB device address of USB switch 30 and sends the modified handshake packet to host 10, which ends the transaction.

[0046] By re-ordering the second token packet ahead of the first data-out packet, reading of USB flash storage block 23 can begin earlier, at time T1 rather than at time T2. This

allows the read data to be ready earlier, so that the second transaction can end sooner. Data throughput can be improved using such packet re-ordering. In an actual system, the read access time can be longer than shown in this simplified diagram, causing a more significant delay that is reduced by re-ordering.

[0047] Due to buffering, packet transmissions from USB switch 30 may be delayed relative to packet reception more than what is shown in Figs. 4, 5. A shift or delay at USB switch 30 may occur but is not shown in the diagrams to improve clarity of understanding basic concepts.

[0048] Figure 6 shows a packet being re-ordered between upstream and downstream queues. The order of packets sent and received from host 10 in Figs. 4, 5 is shown in the upstream queue to the host. Bi-directional queues are shown that have packets in both directions, although an actual implementation may have separate queues for each direction of packet flow. Separate downstream queues for each device may also be used.

[0049] The host first sends the first token packet with the read command to device 1, followed by the data-out packet with the write data. The first handshake packet is then sent to the host.

[0050] Upon receipt of the first handshake packet, the host begins the second transaction with the second token packet with the read command to device 2. The second flash device eventually reads the requested data and sends it back to the host in the data-in packet, which is followed by the second handshake packet that ends the second transaction.

[0051] In hub mode, USB switch 30 simply copies packets from the upstream queue to the downstream queue (or vice-versa) in the same order. However, in single-endpoint mode, packet can be re-ordered to improve performance. In this example, the second token packet is re-ordered in the downstream queue to USB flash storage blocks 22, 23, 24.

[0052] The second token packet that begins access of the second USB flash storage block is placed before the first data-out packet and the first handshake packet. This allows the read of the second USB flash storage block to begin earlier. Physical completion of the data write to the first USB flash storage block is delayed somewhat, but this is usually not a serious performance issue. The downstream queue reflects the packet re-ordering of Fig. 5 to the USB flash storage blocks.

[0053] Figure 7 is a table showing possible re-ordering packet sequences. The second token packet can be reordered to just after the first token packet in sequences of first and second packets where access is not to the same physical memory location. For example, packets are not re-ordered when the first transaction is a read of address M in segment A, and the second transaction is a read or a write to the same address M in segment A. For reads and writes blocks of memory addresses, when any of the address locations overlap, re-ordering is not allowed.

[0054] When the two adjacent transactions are to a different memory segment, or to different, non-overlapping memory address locations in the same segment, then re-ordering is allowed. The vast majority of cases allows packet re-ordering.

[0055] The addresses from the host can be logical block address (LBA). The LBA is translated internally by transaction manager 36 to generate physical addresses that can be on a single chip or across different chips. Two segments could be on the same chip, but the two segments have addresses for two different locations within the chip. The address translation mechanism is governed by consideration of wear-leveling, bad-block management and mem-

ory management.

[0056] Figure 8 is a flowchart of packet re-ordering by the transaction manager. Transaction manager 36 executes re-order process 100. Transaction manager 36 can initially stay in a loop (Fig. 10) waiting for an interrupt caused by requests from the host. When a host interrupt occurs, such as when USB upstream interface 34 detects a new packet from the host on USB bus 18, the packet is loaded into the upstream queue, step 102. Once two or more packets are in the upstream queue, transaction manager examines the packets in the upstream queue, step 104, to find token packets. When the two token packets in a sequence are to the same flash segment and have overlapping memory locations, step 106, then the packets for these adjacent transactions are moved to the downstream queue, step 109, without re-ordering.

[0057] When token packets for adjacent transactions are to different memory segments, or to different, non-overlapping memory locations, step 106, then packets are re-ordered. The second token packet is moved up into the first transaction, step 108. The second token packet can be placed immediately after the first token packet, as shown in Figs. 5, 6 The re-ordered packets are then copied to the down-

stream queue and processed.

[0058] A virtual storage processor operating in conjunction with re-order process 100 can translates all flash storage command, status, address and data requests in the packets from the host. An internal database can be searched for physical endpoint attributes of the USB flash storage blocks. The logical attributes from the host are then translated to physical endpoint attributes associated with the downstream flash storage endpoints. The transaction manager can use this information to determine when memory overlap occurs in step 106.

[0059] Figure 9 is a flowchart of power-on initialization by the USB switch. Initialization process 110 is activated by a reset or by power on of the USB switch. A busy status is reported back to the host by the USB switch, step 112 while process 110 is executing. Downstream devices such as USB flash storage blocks are interrogated by the USB switch to determine their attributes such as storage capacity and USB device addresses, step 114. A database of these downstream attributes is built, step 116.

[0060] The storage capacity of all downstream USB flash storage blocks is aggregated, step 118, by summing the sizes of good blocks for all chips recorded in the memory-map-

ping tables. Since flash memory can wear out after repeated writing and erasing, some bad blocks may exist. Tables for the bad blocks are generated, step 120 and consolidated for all USB flash storage blocks. Wear-leveling routines or tables for these routines are distributed among the USB flash storage blocks to even wear across the USB flash storage blocks, step 122.

[0061] A ready is generated to the upstream host, step 124, once the memory tables are constructed. The configuration reported to the host is for a single pool of memory, rather than the separate USB flash storage blocks queried by the USB switch, step 126. The attributes reported back to the host are composite attributes, such as a size that is the sum of all good blocks of memory, but a speed that is the speed of the slowest downstream memory. Thus memory from several physical USB flash storage blocks can be aggregated and reported to the host as a single memory with the combined capacity of all good blocks in all USB flash storage blocks.

[0062] Figure 10 is a flowchart of high-level operation of the transaction manager. In operating loop 130, transaction manager 36 can initially stay in a loop waiting for an interrupt caused by requests from the host, step 132. When

a host interrupt occurs, such as when USB upstream interface 34 detects a new packet from the host on USB bus 18, the request is passed to the virtual storage processor, step 134. The virtual storage processor translates all flash storage command, status, address and data requests in the packets from the host. An internal database can be searched for physical endpoint attributes of the USB flash storage blocks. The logical attributes from the host are then translated to physical endpoint attributes associated with the downstream flash storage endpoints.

[0063] When a reply is ready from the downstream flash devices, the virtual storage processor can again be consulted to modify the downstream replies. The attributes of the single endpoint memory are used to report back to the host. A single USB device address is used for all replies so that the host only recognizes one USB device at the USB switch. Thus the host sees a single memory rather than the multiple USB flash storage blocks downstream of the USB switch. One virtual endpoint is shown to the host by the USB switch.

[0064] Figure 11 is a flowchart of operation of the virtual storage processor. When the transaction manager sends a request to virtual storage processor 140, any commands for the

flash memory, addresses, status information, or data requests are translated from the request, step 138. The logical address from the request is looked up in a database to find the corresponding or assigned physical address in the USB flash storage blocks, step 142. Using attributes such as storage-block sizes stored with the matching entry or record in the database, logical attributes from the host are translated into physical attributes of the physical USB flash storage blocks, step 144. The one logical request may be spread across several physical USB flash storage blocks, so attributes for each of the USB flash storage blocks are read, perhaps using several lookups. For example, some of the physical USB flash storage blocks may be faster than others.

[0065] Once the physical attributes are known, the flash operations can be performed, step 146. Multiple USB flash storage blocks may be accessed to fulfill the logical request from the host. Accesses to the multiple USB flash storage blocks may be overlapped or interleaved to occur at the same time to improve performance. This interleaving or overlapping is hidden from the host.

[0066] Once the physical USB flash storage blocks have been accessed, a reply to the host is generated, step 146. When

packets are re-ordered, this reply may be generated before the physical operation is complete, such as for a flash write.

[0067] Data can be stored in the USB flash storage blocks in several arrangements. For example, data can be stored conventionally in a linear fashion across the USB flash storage blocks. Alternatively, data can be mirrored to provide redundant storage, such as for a redundant array of independent disks (RAID) system. For data mirroring, the physical flash storage is partitioned into two equal-size logical segments. Data is written to both segments and can be read back from either segment.

[0068] Figure 12 shows a data access routine in a mirrored flash system. Mirrored flash-access routine 150 is called by the transaction manager to read or write data on the physical USB flash storage blocks when data mirroring has been enabled. The type of operation, read or write, is detected from the request, step 152. For writes (or erase) operations, the data is written to both logical segments, step 160. Thus the new data is stored in both logical segments, providing redundant storage locations. Access records can be updated, step 162, before control is passed back to the transaction manager.

[0069] For read operations, the requested data is read from the active logical segment, step 154. One of the two mirrored logical segments is designated as the currently active segment from which reads are performed. The currently active segment could remain the same logical segment for a long time, or the active segment could change periodically or when an error occurs.

[0070] When the read was performed without errors, step 156, then the access records can be updated, step, 162, and control is passed back to the transaction manager. Reading errors might be detected by parity checks or by a physical flash memory device not responding. Errors may be caused by an earlier over-write, over-erasure or by a defect in a page or block When an error occurred during reading, step 156, the other or backup logical segment is read from, step 158. Access records can be updated, step, 162, and control is passed back to the transaction manager. The backup storage segment allows for instant data recovery in case of failure.

[0071] Figure 13 is a flowchart of flash-memory access using data striping. The USB flash storage blocks are partitioned into multiple segments of equal size. Each data item can be stored (as a stripe) across the multiple segments. Ide-

ally, the segments are chosen to be on as many different physical USB flash storage blocks as possible so that a physical device failure affects only a small portion of any data item.

[0072] One of the multiple segments is designates as a parity segment, and stores parity or other error-correcting code for the other segments. When a failure occurs in one of the segments, the parity information can be used to re-construct the data item. For example, an 8-bit data item could be stored as a stripe across 8 separate flash-memory chips, with one bit stored per chip. A ninth flash-memory chip stores the parity bits. If one of the 8 flash-memory chips fails, the parity bit can be used to deter-mine the missing bit from the failed flash-memory chip.

[0073] When the transaction manager access data in a striped storage system, striped access routine 170 is called. The type of access, read or write/erase, is determined from the request, step 172. For write/erase, the data from the host is unpacked or split into portions that are written to separate segments, step 176. The unpacked portions of the write data are then written to the multiple segments, step 178. For example, a data byte could be unpacked into 8 one-bit portions that are written to 8 different flash

memory chips. The parity data is also generated during unpacking and written to the parity segment.

[0074] For a read operation, the data is read from the multiple segments, step 174. If any errors are detected, step 180, then the missing data is reconstructed using the remaining data and the parity information from the parity segment, step 182. The read data is then re-packed into a single data item from the multiple portions of data read from the multiple segments, step 184. The parity information is removed before sending the data item back to the transaction manager and on to the host.

[0075] Figures 14A-C show various arrangements of data stored in the USB flash storage blocks. In Fig. 14A, data is arranged in a conventional linear arrangement. Each N-bit data item, such as an 8-bit byte or a 32-bit word, is stored in a memory location that is physically on one USB flash storage device. A total of M data items are stored, with some of the data items being stored on different USB flash storage devices. When a failure occurs, such as a flash-memory chip failing to return data, the entire data item is usually lost. However, other data items stored on other physical flash-memory chips can be read without errors.

[0076] In Fig. 14B, data is striped across M flash-storage segments. Each data item is stored in one of the M flash-storage segments. For example, an N-bit data item consists of bits 11, 12, 13, ... 1N. The data item has all bits 11-1N stored in segment 1. Another data item consists of bits 21,22, 23, ...2N. The data item has all bits 21-2N stored in segment 2. Data items can fill up one segment before starting to fill the next segment, or data items could be interleaved across the segments.

[0077] In Fig. 14C, data striping is performed across multiple storage segments with parity. The USB flash storage blocks are partitioned into N+1 segments. The N segments are equal size, and the parity segment is sufficiently large in size to hold parity or error-correcting code (ECC) for the other N segments.

[0078] Each data item is divided into N portions with each portion stored on a different one of the N segments. The parity bit or ECC for the data item is stored in the parity segment. For example, an N-bit data item consists of bits 11, 12, 13, ... 1N. The data item has bit 11 stored in segment 1, bit 12 stored in segment 2, bit 13 stored in segment 3, ...and bit N stored in segment N. The parity bit, bits, or ECC is stored in the parity segment as bit or bits 1P.

[0079] In the diagram, each data item is stored across all segments as a vertical stripe. If one segment fails, most of the data item remains intact, allowing for recovery using the parity bit.

[0080] ALTERNATE EMBODIMENTS

[0081] Several other embodiments are contemplated by the inventors. For example different numbers and arrangements of USB flash storage blocks can connect to the USB switch. Rather than use USB buses, other serial buses such as PCI Express, ExpressCard, Firewire (IEEE 1394), serial ATA, serial attached small-computer system interface (SCSI), etc. The mode logic could sense the state of a pin only at power-on rather than sense the state of a dedicated pin. A certain combination or sequence of states of pins could be used to initiate a mode change, or an internal register such as a configuration register could set the mode. For example, when PCI Express is used, additional pins for the PCI Express interface can be added or substituted for the USB differential data pins. PCI express pins include a transmit differential pair PET+, PET-, and a receive differential pair PER+, PER- of data pins. A multi-bus-protocol chip could have an additional personality pin to select which serial-bus interface to use, or could have pro-

grammable registers. ExpressCard has both the USB and the PCI Express bus, so either or both buses could be present on an ExpressCard device.

[0082] The transaction manager and its controllers and functions can be implemented in a variety of ways. Functions can be programmed and executed by a CPU or other processor, or can be implemented in dedicated hardware, firmware, or in some combination. Many partitionings of the functions can be substituted.

[0083] Wider or narrower data buses and flash-memory chips could be substituted, such as with 16 or 32-bit data channels. Alternate bus architectures with nested or segmented buses could be used internal or external to the controller. Two or more internal buses can be used in the USB switch to increase throughput. More complex switch fabrics can be substituted for the internal bus.

[0084] Data striping can be done in a variety of ways, as can parity and error-correction code (ECC). Packet re-ordering can be adjusted depending on the data arrangement used to prevent re-ordering for overlapping memory locations. The USB switch can be integrated with other components or can be a stand-alone chip.

[0085] Errors may be detected through two-dimensional error

checking and correction. Each storage segment, including the parity segment, has a page-based ECC. When a segment page is read, bad bits can be detected and corrected according to the strength of the ECC code, such as a Reed-Solomon code. In addition, the flash storage segments form a stripe with parity on one of the segments. Assuming there are four storage segments F(1), F(2), F(3), F(4) and one parity segment F(P). These five segments form even parity stored on F(P). Each segment has its own independent ECC to do the first level of error detection and correction. If the first level ECC fails on segment F(2), the corresponding striping bit information on F(1), F(3), F(4) and F(P) are sufficient to recover what bit information should be on F(2). The two levels of ECC form a two-dimension error checking and correction.

[0086] The abstract of the disclosure is provided to comply with the rules requiring an abstract, which will allow a searcher to quickly ascertain the subject matter of the technical disclosure of any patent issued from this disclosure. It is submitted with the understanding that it will not be used to interpret or limit the scope or meaning of the claims. 37 C.F.R. Sect. 1.72(b). Any advantages and benefits described may not apply to all embodiments of the inven-

tion. When the word "means" is recited in a claim element, Applicant intends for the claim element to fall under 35 USC Sect. 112, paragraph 6. Often a label of one or more words precedes the word "means". The word or words preceding the word "means" is a label intended to ease referencing of claims elements and is not intended to convey a structural limitation. Such means-plus-function claims are intended to cover not only the structures described herein for performing the function and their structural equivalents, but also equivalent structures. For example, although a nail and a screw have different structures, they are equivalent structures since they both perform the function of fastening. Claims that do not use the word "means" are not intended to fall under 35 USC Sect. 112, paragraph 6. Signals are typically electronic signals, but may be optical signals such as can be carried over a fiber optic line.

[0087] The foregoing description of the embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the inven-

tion be limited not by this detailed description, but rather by the claims appended hereto.